AMENDMENTS TO THE SPECIFICATION

Please replace the paragraph on page 10, lines 18-22, with the following amended paragraph:

A1

Referring to Fig. 3, in an intermediate representation, a method is represented by a static single assignment (SSA) graph [[140]] 340 embedded in the control flow graph (Fig. 4). The structure of the SSA graph [[140]] 340 of Fig. 3, the structure of the control flow graph of Fig. 4, and the relationship between the SSA graph and the control flow graph will be described.

Please replace the paragraph on page 10, lines 27-33, with the following amended paragraph:

A2

In Fig. 3, an exemplary static single assignment (SSA) graph 340 for a load operation is shown. The SSA graph 340 has nodes 342-1, 342-2, 342-3 and 342-4, referred to as values, that represent individual operations. Referring also to Fig. 5, a generalized form for a value 342 is shown. The ovals [[342]] in Fig. 3 represent the nodes or SSA values, and the boxes [[344,]] represent blocks in the control flow graph. A value may have one or more inputs, which are the result of previous operations, and has a single result, which can be used as the input for other values.

Please replace the paragraph on page 11, lines 13-23, with the following amended paragraph:

A3

The intermediate representation breaks out into separate operations the required run-time checks associated with the programming language, such as Java™, used to implement the method. Compiler 58 has individual operations representing null checks, bounds checks, and other kinds of run-time checks. These operations cause a run-time exception if their associated check fails. A value 342 representing a run-time check produces a result that has no representation in the generated machine code. However, other values [[342]] that depend on the run-time check take its result as an input so as to ensure that these values are scheduled after the run-time check. Representing the run-time checks as distinct operations allows compiler 58 to apply

A3

optimizations to the checks, such as using common subexpression elimination on two null checks of the same array.

Please replace the paragraph on page 13, lines 1-12, with the following amended paragraph:

A4

Each control flow graph 460 has a single entry block 464, a single normal exit block [[466]] 468, and a single exception exit block [[468]] 466. The entry block 464 includes the values representing the input arguments of the method. The normal exit block [[466]] 468 includes the value representing the return operation of the method. The exceptional exit block [[468]] 466 represents the exit of a method that results when an exception is thrown that is not caught within the current method. Because many operations can cause run-time exceptions in Java™ and these exceptions are not usually caught within the respective method in which the exception occurs, many blocks of the graph may have an exception edge to the exception exit block [[468]] 466. Blocks B1 and B2, 462-2 and 462-3, respectively, form a loop. Block B1 462-2 has an exception exit and is connected to the exception exit block [[468]] 466. Block B2 462-3 is connected to the normal exit block [[466]] 468.